MERN STACK PROJECT

Book-Barter-System Website

Done By:

Advaith Dinesan Pudussery (1HK22CS008)
Ihsan Rafeeque Sainudheen (1HK22CS055)
Jithin P Joji (1HK22CS051)
Balasubramani B (1HK22CS026)
Vijay Kumar (1HK22EC180)

Table of Contents

Sl no.	Content	Page No.
1	Introduction	1
2	Problem Statement & Objective	2
3	Workflow	3
4	Source Code	5
5	Output	14
6	Future Improvements	17
7	Conclusions	19
8	References	20

Introduction

The Book Barter System is a MERN (MongoDB, Express.js, React, Node.js) stack-based web application designed to facilitate peer-to-peer book trading among users. This platform allows users to register and log in securely using encrypted credentials via bcrypt hashing. Once authenticated, users can list their books for trade, browse available books shared by others, and initiate contact to exchange books.

Key Features:

- **User Authentication:** A secure login and registration system implemented using bcrypt and JWT ensures that only authorized users can add or trade books.
- **OpenLibrary API Integration:** When a user wishes to add a book, they can search using book titles or authors. The app fetches detailed book data from the OpenLibrary API, including the title, author, genre, page count, and cover image.
- **Book Management:** Users can add books to the collection, which are then stored in a MongoDB database. Each book entry contains essential metadata along with the owner's contact email.
- **Browse & Discover:** The home screen displays a dynamic listing of all books available for trade from different users, allowing anyone to explore the shared library.
- **Book Details & Trade:** On clicking a book, users are taken to a detailed view page. If they wish to initiate a trade, a "Ready to Trade" button reveals the email address of the book owner, enabling direct communication.
- **Availability Status:** Each book record has an available flag to indicate whether the book is open for trade, ensuring efficient inventory control and transparency.

This project demonstrates full-stack development skills by integrating front-end interaction, RESTful APIs, external API consumption, MongoDB schema design, and secure authentication mechanisms. It aims to promote a community-driven culture of knowledge sharing and sustainability by encouraging users to reuse and circulate books.

Problem Statement

In today's digital age, many students and readers accumulate books that are rarely reused or recycled after their initial use. Simultaneously, others seek affordable access to similar books but often face financial constraints or limited availability. Traditional book exchange systems are mostly informal, localized, and lack structure or accessibility across wider communities.

There is a clear lack of a centralized, user-friendly platform that enables book owners to share, trade, or exchange books with others in a secure and efficient manner. Existing online marketplaces focus primarily on selling or donating books rather than encouraging peer-to-peer bartering.

Objective

To address this gap, we propose the Book Barter System — a web-based platform that allows users to:

- Register and authenticate securely
- Add their books to a shared online collection using reliable metadata from the OpenLibrary API
- Explore books listed by other users
- Initiate trades by directly contacting book owners willing to exchange

This system promotes a culture of sustainable reading, cost-efficiency, and community collaboration by enabling users to exchange books instead of purchasing new ones.

Workflow of Book Barter System

The Book Barter System follows a well-defined workflow from user registration to book trading. Below is the step-by-step flow of how users interact with the system:

1. User Registration / Login

- New users register using their name, email, and password.
- Passwords are hashed using bcrypt for security.
- Existing users can log in using their email and password.
- Upon successful authentication, users receive access to the platform.

2. Home Screen – Book Listing

- After login, users are redirected to the Home Page, which displays all books available in the system.
- Books are fetched from MongoDB and displayed with details such as title, author, and cover image.

3. Add a Book

- Users click on the “Add Book” button.
- A search feature allows users to look up books using the OpenLibrary API.
- Once a book is selected, users fill in additional details such as:
    - Genre
    - Condition
    - Description
- Book information along with the user’s email and ID is saved to MongoDB.

4. View Book Details

- On clicking a specific book in the home screen, a Book Detail Page opens.
- This page shows:
    - Full book details
    - Description
    - Owner's contact email (only visible after clicking “Ready to Trade”)

5. Ready to Trade

- When a user is interested in trading a book, they click “Ready to Trade”.
- The system reveals the email address of the book’s owner so they can initiate a trade directly.

6. Book Availability

- Each book has an `available` flag.
- Once a book is traded, its status can be updated to unavailable to prevent further requests.

Source Code

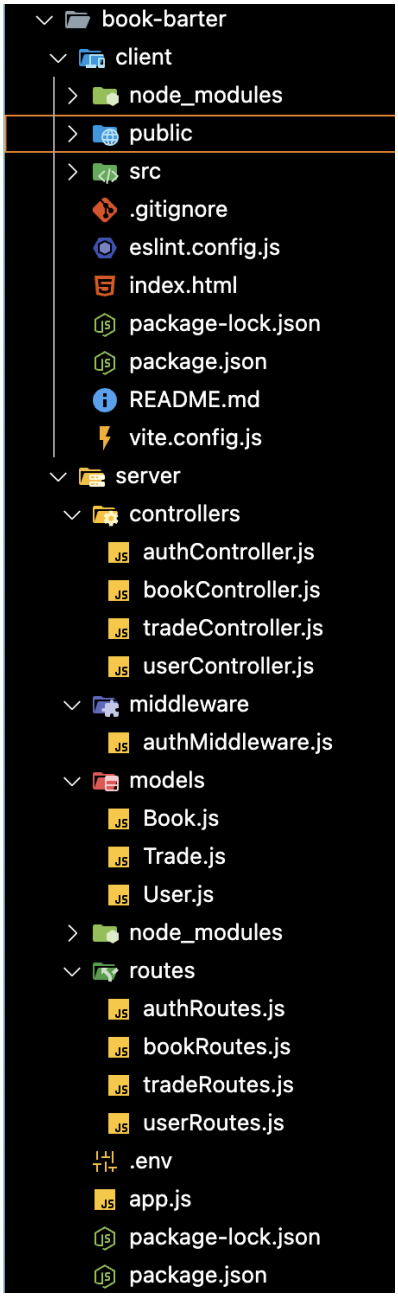


fig.1) Project Directory

```jsx
import React, { useState } from 'react';
import axios from 'axios';

const AddBook = () => {
  const [search, setSearch] = useState('');
  const [results, setResults] = useState([]);
  const token = localStorage.getItem('token');

  const searchBooks = async () => {
    try {
      const res = await axios.get(`https://openlibrary.org/search.json?q=${search}`);
      setResults(res.data.docs.slice(0, 5));
    } catch (error) {
      console.error('OpenLibrary search failed:', error);
    }
  };

  const handleAddBook = async (book) => {
    try {
      await axios.post('http://localhost:3030/api/books/add', {
        title: book.title,
        author: book.author_name ? book.author_name.join(', ') : 'Unknown',
        genre: book.subject ? book.subject[0] : 'General',
        condition: 'Good',
        description: book.first_sentence ? book.first_sentence[0] : 'No description available',
        pageCount: book.number_of_pages_median || 100,
        imageUrl: `https://covers.openlibrary.org/b/id/${book.cover_i}-M.jpg`,
        email: localStorage.getItem('userEmail') || 'No email available',
      }, {
        headers: {
          Authorization: `Bearer ${token}`,
        }
      });
      alert('Book added!');
    } catch (err) {
      console.error('Error adding book:', err);
    }
  };

  return (
    <div className="add-book-container">
      <h2>🔍 Search & Add Book</h2>
      <div className="add-book-searchbar">
        <input
          type="text"
          value={search}
          onChange={(e) => setSearch(e.target.value)}
          placeholder="Search book from OpenLibrary"
        />
        <button onClick={searchBooks}>Search</button>

      </div>

      <div className="add-book-grid">
        {results.map((book, index) => (
          <div key={index} className="add-book-card">
            <img
              src={book.cover_i
                ? `https://covers.openlibrary.org/b/id/${book.cover_i}-M.jpg`
                : 'https://via.placeholder.com/128x180?text=No+Image'}
              alt={book.title}
            />
            <div className="add-book-content">
              <h4>{book.title}</h4>
              <p>{book.author_name ? book.author_name.join(', ') : 'Unknown Author'}</p>
              <button onClick={() => handleAddBook(book)}>Add</button>
            </div>
          </div>
        ))}
      </div>
    </div>
  );
};

export default AddBook;
```

fig.2) AddBooks.jsx

```jsx
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import BookSearch from './BookSearch';
import BookDetails from './BookDetails';
import Signup from './Signup';
import Login from './Login';
import AddBook from './AddBook';

const ProtectedRoute = ({ children }) => {
  const token = localStorage.getItem('token');
  return token ? children : <Navigate to="/login" />;
};

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/signup" element={<Signup />} />
        <Route path="/login" element={<Login />} />

        <Route
          path="/"
          element={
            <ProtectedRoute>
              <BookSearch />
            </ProtectedRoute>
          }
        />
        <Route
          path="/book/:id"
          element={
            <ProtectedRoute>
              <BookDetails />
            </ProtectedRoute>
          }
        />
        <Route
          path="/add-book"
          element={
            <ProtectedRoute>
              <AddBook />
            </ProtectedRoute>
          }
        />
      </Routes>
    </Router>
  );
}

export default App;

```

fig.3) App.jsx

```jsx
import './App.css';

const BookDetails = () => {
  const { id } = useParams();
  const [book, setBook] = useState(null);
  const [showEmail, setShowEmail] = useState(false); // state for toggling owner's email

  useEffect(() => {
    fetch(`http://localhost:3030/api/books/${id}`, {
      headers: {
        Authorization: `Bearer ${localStorage.getItem('token')}`
      }
    })
      .then(res => res.json())
      .then(data => setBook(data))
      .catch(err => console.error('Error:', err));
  }, [id]);

  if (!book) return <p style={{ color: 'white' }}>Loading...</p>;

  const ownerEmail = book.owner?.email || 'Email not available';

  return (
    <div className="book-details-container">
      <div className="left-box">
        <img
          src={book.imageUrl || 'https://via.placeholder.com/300x400?text=No+Image'}
          alt={book.title}
          className="book-image"
        />
        <div className="info-box">
          <p><strong>Book Name:</strong> {book.title}</p>
          <p><strong>Author:</strong> {book.author}</p>
          <p><strong>Genre:</strong> {book.genre}</p>
          <p><strong>Condition:</strong> {book.condition}</p>
          <p><strong>Page Count:</strong> {book.pageCount}</p>
          <p><strong>Owner:</strong> {book.owner?.name || 'Unknown'}</p>
        </div>
      </div>

      <div className="right-box">
        <div className="description-box">
          <p>{book.description || 'No description available.'}</p>
        </div>

        <button className="trade-button" onClick={() => setShowEmail(true)}>
          Ready to Trade
        </button>

        {showEmail && (
          <p style={{ marginTop: '10px', color: 'black' }}>
            Contact Email: <strong>{ownerEmail}</strong>
          </p>
        )}
      </div>
    </div>
  );
};

export default BookDetails;
```

fig.4) BookDetails.jsx

```jsx
import { useEffect, useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import axios from 'axios';

const BookSearch = () => {
  const [query, setQuery] = useState('');
  const [books, setBooks] = useState([]);
  const token = localStorage.getItem('token');
  const navigate = useNavigate();
  const logout=()=>{
    localStorage.removeItem('token');
    localStorage.removeItem('email');
    navigate('/login');
  }
  const fetchBooks = async () => {
    try {
      const response = await axios.get('http://localhost:3030/api/books/', {
        headers: { Authorization: `Bearer ${token}` },
      });
      setBooks(response.data);
    } catch (error) {
      console.error('Error fetching books:', error);
    }
  };

  useEffect(() => {
    fetchBooks();
  }, []);

  const filteredBooks = books.filter((book) =>
    book.title.toLowerCase().includes(query.toLowerCase()) ||
    (book.author && book.author.toLowerCase().includes(query.toLowerCase()))
  );

  return (
    <div className="container">
      <h2>📚 My Book List</h2>
      <div className="search-bar">
        <input
          type="text"
          value={query}
          onChange={(e) => setQuery(e.target.value)}
          placeholder="Search by title or author"
        />
        <button onClick={() => navigate('/add-book')}>➕ Add Book</button>
        <button onClick={logoutbv}>LogOut</button>
      </div>

      <div className="card-grid">
        {filteredBooks.map((book, index) => {
          const coverImage = book.imageUrl
            ? book.imageUrl
            : `https://via.placeholder.com/128x180?text=No+Image`;

          return (
            <Link
              to={`/book/${book._id}`}
              key={index}
              style={{ textDecoration: 'none', color: 'inherit' }}
            >
              <div className="card">
                <img src={coverImage} alt={book.title} />
                <div className="card-content">
                  <h4>{book.title}</h4>
                  <p>{book.author || 'Unknown Author'}</p>
                  <button>View</button>
                </div>
              </div>
            </Link>
          );
        })}
      </div>
    </div>
  );
};
```

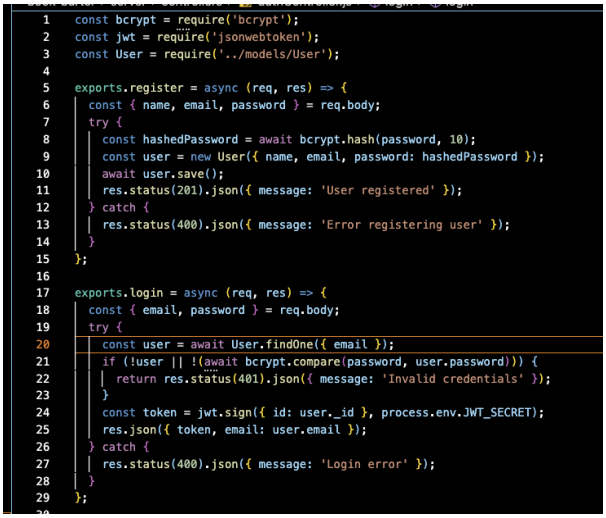
fig.5) BookSearch.jsx

```jsx
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';

function Login() {
  const [form, setForm] = useState({ email: '', password: '' });
  const navigate = useNavigate();

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const res = await axios.post('http://localhost:3030/api/auth/login', form);
      const token = res.data.token;
      const email = res.data.email;

      localStorage.setItem('token', token);
      localStorage.setItem('email', email);

      alert('Login successful');
      navigate('/');
    } catch (err) {
      alert(err.response?.data?.message || 'Login failed');
    }
  };

  return (
    <div className="login-container">
      <h2>📚 Book App Login</h2>
      <form id="loginForm" onSubmit={handleSubmit}>
        <input
          type="email"
          name="email"
          placeholder="Email"
          onChange={handleChange}
          required
        />
        <input
          type="password"
          name="password"
          placeholder="Password"
          onChange={handleChange}
          required
        />
        <button type="submit">Login</button>
      </form>
      <p className="note">
        Don't have an account? <a href="/signup">Register</a>
      </p>
    </div>
  );
}

export default Login;
```

fig.6) Login.jsx

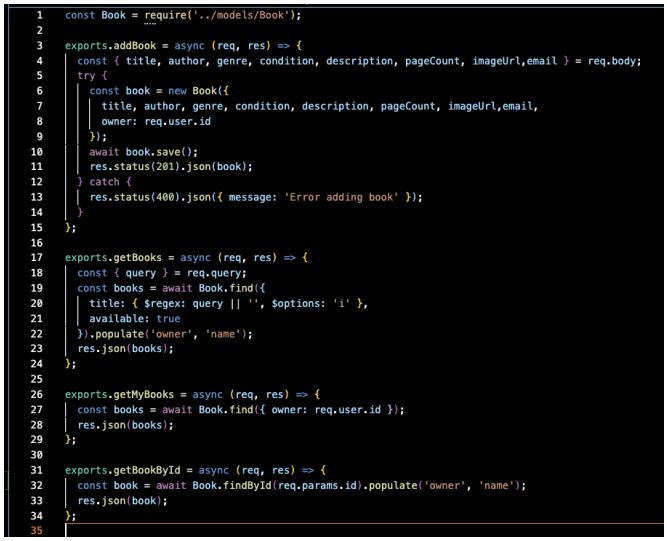
```jsx
// src/components/Signup.js
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

function Signup() {
  const [form, setForm] = useState({ name: '', email: '', password: '' });
  const navigate = useNavigate();

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await axios.post('http://localhost:3030/api/auth/register', form);
      alert('Signup successful');
      navigate('/login');
    } catch (err) {
      alert(err?.response?.data?.message || 'Signup failed');
    }
  };

  return (
    <form onSubmit={handleSubmit} className="signup-form">
      <h2 className="signup-title">Signup</h2>
      <input name="name" className="signup-input" placeholder="Name" onChange={handleChange} required />
      <input name="email" className="signup-input" placeholder="Email" type="email" onChange={handleChange} required />
      <input name="password" className="signup-input" placeholder="Password" type="password" onChange={handleChange} required />
      <button type="submit" className="signup-button">Register</button>
    </form>
  );
}

export default Signup;
```

fig.7) SignUp.jsx



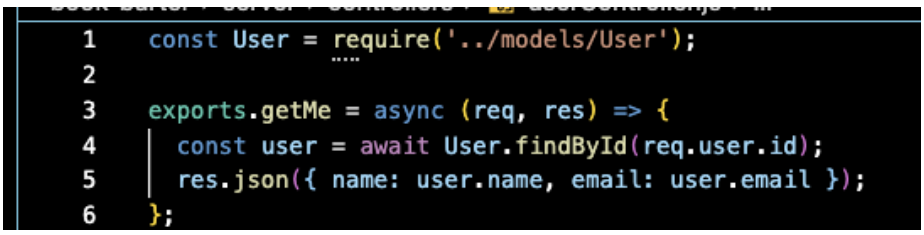
```
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const User = require('../models/User');

exports.register = async (req, res) => {
  const { name, email, password } = req.body;
  try {
    const hashedPassword = await bcrypt.hash(password, 10);
    const user = new User({ name, email, password: hashedPassword });
    await user.save();
    res.status(201).json({ message: 'User registered' });
  } catch {
    res.status(400).json({ message: 'Error registering user' });
  }
};

exports.login = async (req, res) => {
  const { email, password } = req.body;
  try {
    const user = await User.findOne({ email });
    if (!user || !(await bcrypt.compare(password, user.password))) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }
    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET);
    res.json({ token, email: user.email });
  } catch {
    res.status(400).json({ message: 'Login error' });
  }
};
```

fig.8) AuthController.js



```
const Book = require('../models/Book');

exports.addBook = async (req, res) => {
  const { title, author, genre, condition, description, pageCount, imageUrl,email } = req.body;
  try {
    const book = new Book({
      title, author, genre, condition, description, pageCount, imageUrl,email,
      owner: req.user.id
    });
    await book.save();
    res.status(201).json(book);
  } catch {
    res.status(400).json({ message: 'Error adding book' });
  }
};

exports.getBooks = async (req, res) => {
  const { query } = req.query;
  const books = await Book.find({
    title: { $regex: query || '', $options: 'i' },
    available: true
  }).populate('owner', 'name');
  res.json(books);
};

exports.getMyBooks = async (req, res) => {
  const books = await Book.find({ owner: req.user.id });
  res.json(books);
};

exports.getBookById = async (req, res) => {
  const book = await Book.findById(req.params.id).populate('owner', 'name');
  res.json(book);
};
```

fig.9) BookController.js



```
const User = require('../models/User');

exports.getMe = async (req, res) => {
  const user = await User.findById(req.user.id);
  res.json({ name: user.name, email: user.email });
};
```

fig.10) UserController.js

```jwt
const jwt = require('jsonwebtoken');
module.exports = (req, res, next) => {
  const token = req.headers.authorization?.split(' ')[1];
  if (!token) return res.status(401).json({ message: 'No token provided' });
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch {
    res.status(403).json({ message: 'Invalid token' });
  }
};
```

fig.11) AuthMiddleware.js

```
const mongoose = require('mongoose');
const bookSchema = new mongoose.Schema({
  title: String,
  author: String,
  genre: String,
  condition: String,
  description: String,
  pageCount: Number,
  imageUrl: String,
  email: String, // Added email field
  owner: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  available: { type: Boolean, default: true }
});
module.exports = mongoose.model('Book', bookSchema);
```

fig.12) Model/Books.js

```
const mongooses = require('mongoose');
const userSchema = new mongooses.Schema({
  name: String,
  email: { type: String, unique: true },
  password: String
});
module.exports = mongooses.model('User', userSchema);
```

fig.13) Model/user.js

```
const express = require('express');
const { register, login } = require('../controllers/authController');
const router = express.Router();

router.post('/register', register);
router.post('/login', login);

module.exports = router;
```

fig.14) AuthRoute.js

```
1  const express = require('express');
2  const { addBook, getBooks, getMyBooks, getBookById } = require('../controllers/bookController');
3  const auth = require('../middleware/authMiddleware');
4  const router = express.Router();
5
6  router.post('/add', auth, addBook);
7  router.get('/', getBooks);
8  router.get('/mine', auth, getMyBooks);
9  router.get('/:id', getBookById);
10
11 module.exports = router;
```

fig.15) bookRoute.js

```
1  const express = require('express');
2  const mongoose = require('mongoose');
3  const dotenv = require('dotenv');
4  const cors = require('cors');
5
6  const authRoutes = require('./routes/authRoutes');
7  const bookRoutes = require('./routes/bookRoutes');
8  const tradeRoutes = require('./routes/tradeRoutes');
9  const userRoutes = require('./routes/userRoutes');
10
11 dotenv.config();
12
13 const app = express();
14 app.use(cors());
15 app.use(express.json());
16
17 mongoose.connect(process.env.MONGO_URI, {
18   useNewUrlParser: true,
19   useUnifiedTopology: true
20 }).then(() => console.log('MongoDB connected'));
21
22 app.use('/api/auth', authRoutes);
23 app.use('/api/books', bookRoutes);
24 app.use('/api/trades', tradeRoutes);
25 app.use('/api/users', userRoutes);
26
27 const PORT = process.env.PORT || 3030;
28 app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
29
```

fig.15) App.js

```
1  MONGO_URI=mongodb://localhost:27017/bookbarter
2  JWT_SECRET=5647ydhsjiwkkh5647ydhsjiwkkh
3
```

fig.16) .env

Output

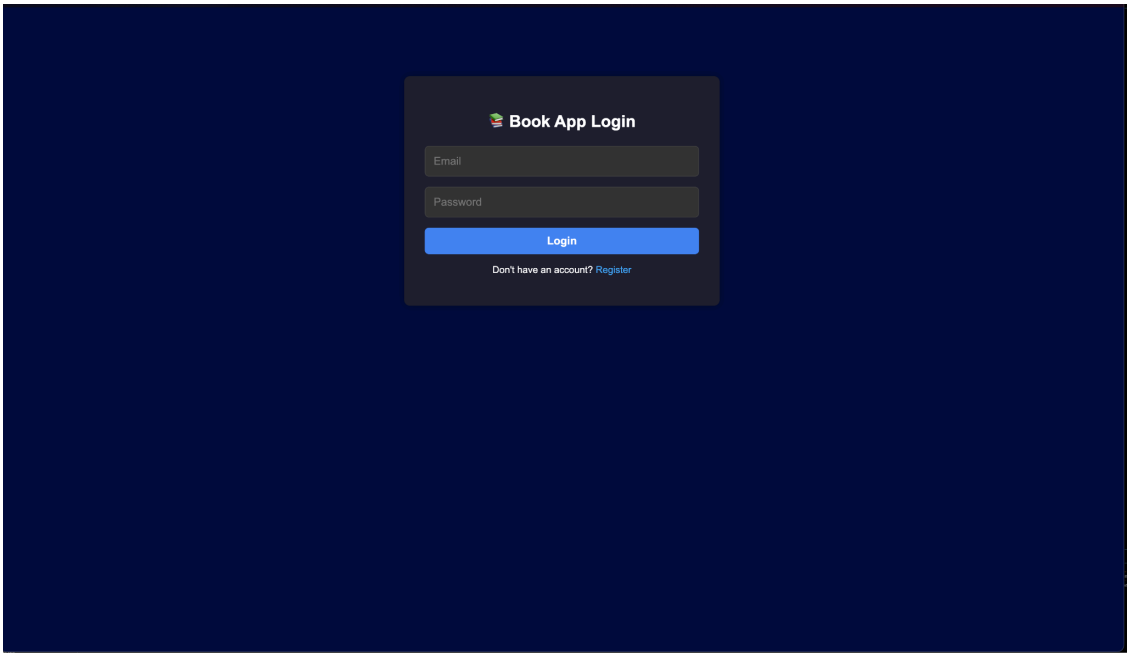


fig.1) Signup Page

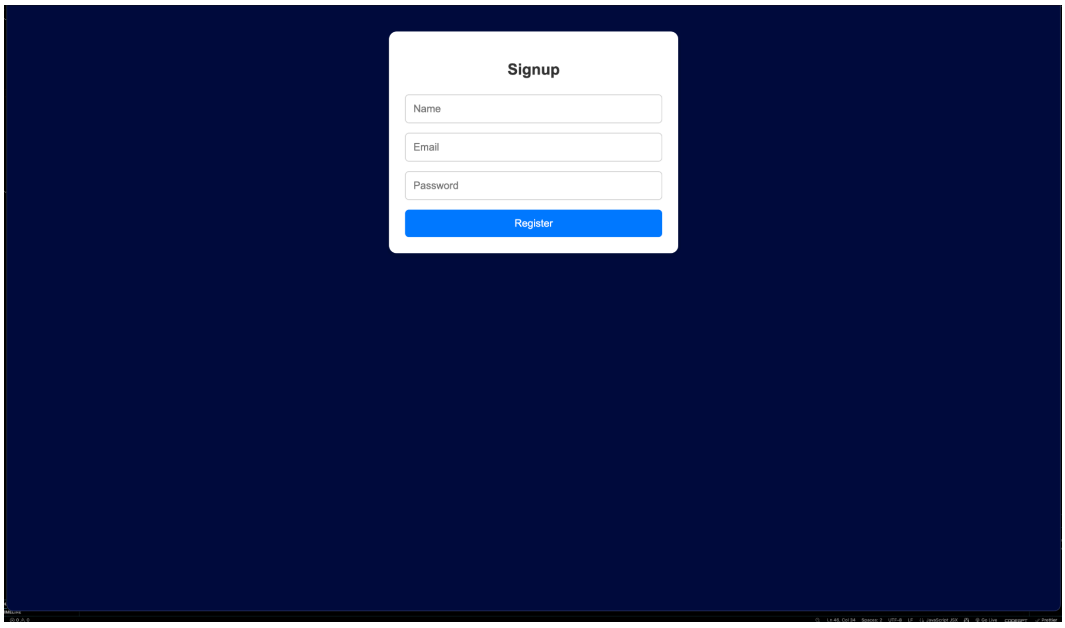


fig.2) Login Page

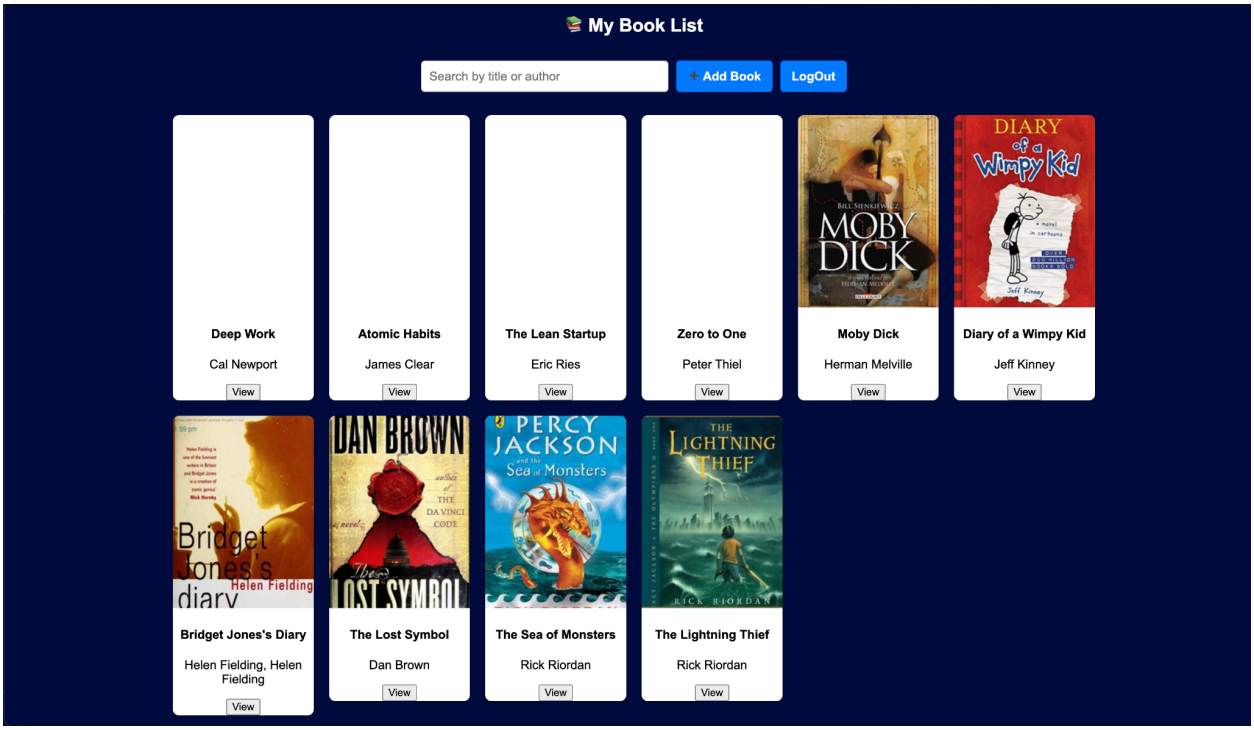


fig.3) Home Page

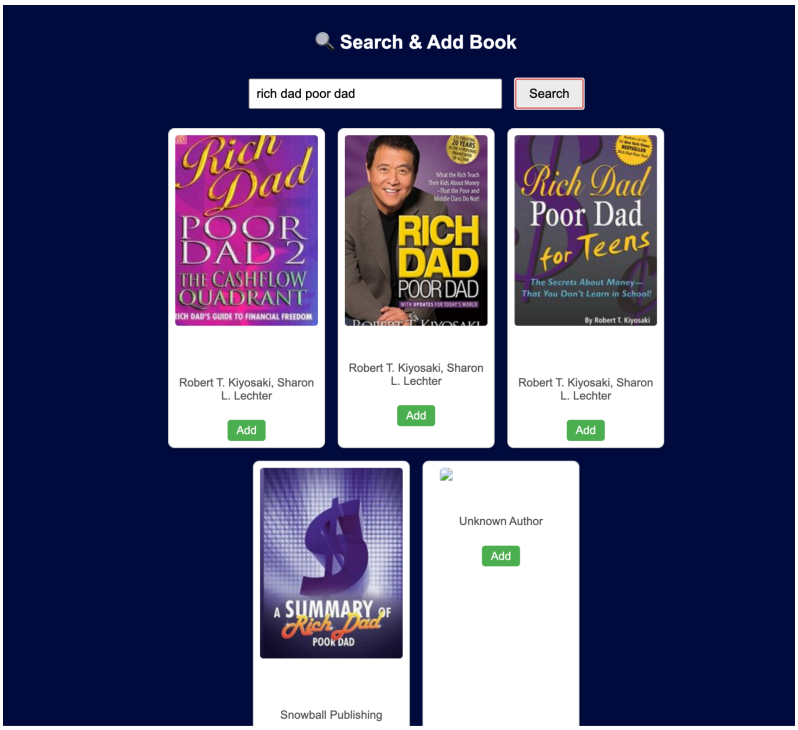


fig.4) AddBooks Page

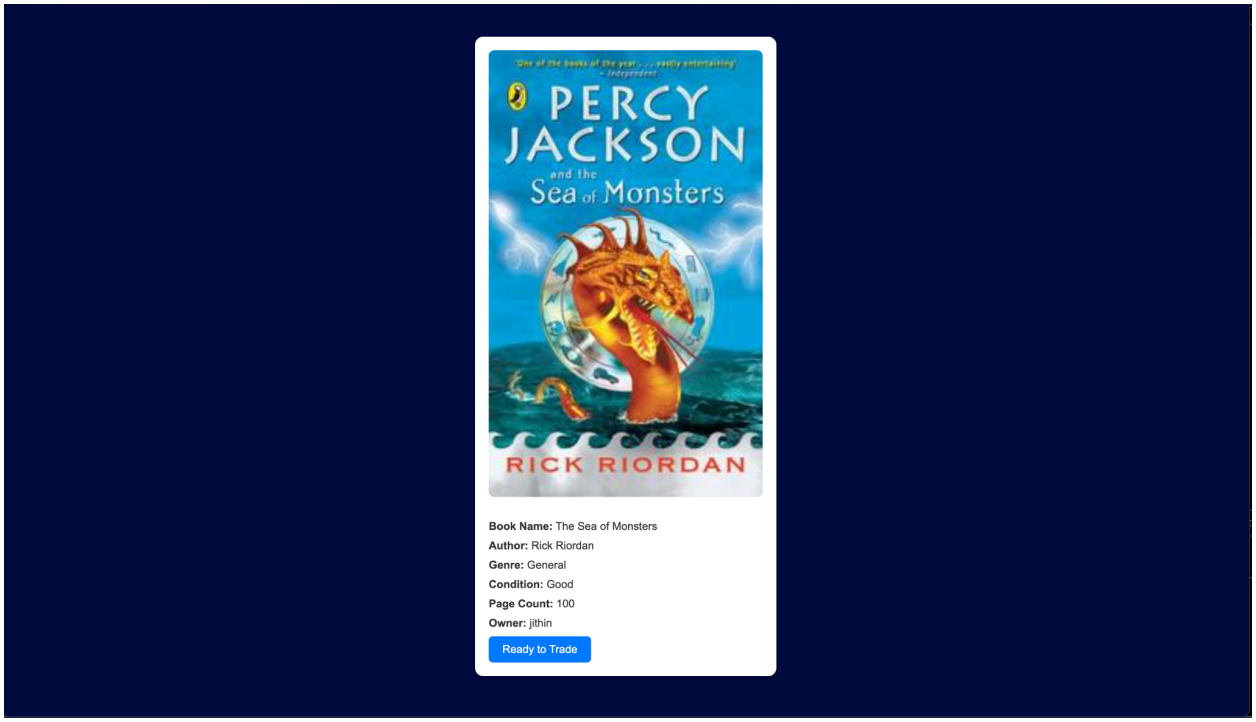


fig.5) BookDetail Page

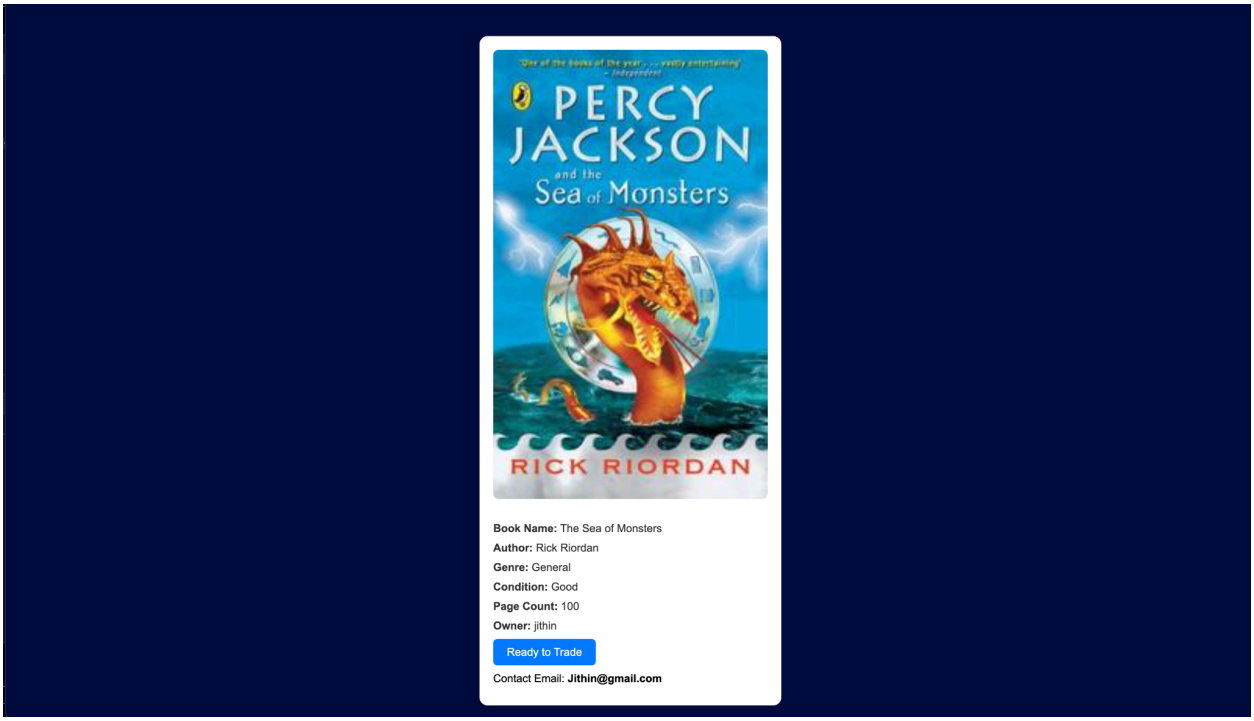


fig.6) BookDetail Page on clicking Ready to Trade

Future Improvements

While the current version of the **Book Barter System** allows users to list and discover books and directly contact owners via email for trade, the system can be significantly enhanced in future iterations to provide a more robust and user-friendly trading experience.

1. Structured Trade Request System

- Instead of directly revealing the owner's email when “Ready to Trade” is clicked, the platform can implement a **trade request mechanism**.
- Users can send a **formal trade request** where they select one of their own books to offer in exchange.
- The owner of the requested book receives a **notification or in-app message** showing:
    - The book the other user wants
    - The book being offered in return
    - Options to **Accept** or **Decline** the request

2. Notification System

- Implement a **notification system** for trade requests using:
    - Email notifications
    - In-app alerts or dashboards for incoming/outgoing requests
- Users can view the status of their requests: *Pending*, *Accepted*, *Declined*

3. Chat or Messaging System

- Enable in-platform messaging so users can **negotiate** and discuss the trade before confirming.
- This keeps communication within the app, improving privacy and user experience.

4. Trade History & Tracking

- Maintain a **trade history** for users to view previous trades.
- Include features like:
    - Trade status
    - Date of exchange

- Feedback or rating system post-trade

5. Book Availability Auto-Update

- Once a trade is accepted, the system should **automatically mark both books as unavailable**, preventing duplicate requests.

Conclusion

The **Book Barter System** successfully demonstrates how technology can be leveraged to promote sustainable reading habits and build a community-driven platform for book sharing and exchange. By combining the power of the MERN stack with secure user authentication and external API integration (OpenLibrary), this system enables users to list, discover, and initiate trades of books efficiently.

The platform not only simplifies the process of finding and sharing books but also encourages collaboration among users without monetary transactions. Through the implementation of core features like login, book listing, detailed viewing, and a “Ready to Trade” function, the system lays a solid foundation for a peer-to-peer book exchange ecosystem.

While the current version provides the essential functionality, several future improvements — such as a structured trade system, in-app messaging, and notification features — can significantly enhance the user experience and scalability of the platform.

Overall, this project reflects a practical application of full-stack web development and problem-solving to address a real-world need in the field of education and reading culture.

References

1. **MongoDB Documentation**
   https://www.mongodb.com/docs/

2. **Express.js Documentation**
   https://expressjs.com/

3. **React Official Documentation**
   https://reactjs.org/docs/getting-started.html

4. **Node.js Documentation**
   https://nodejs.org/en/docs/

5. **OpenLibrary API Documentation**
   https://openlibrary.org/developers/api

6. **bcrypt for Node.js**
   https://www.npmjs.com/package/bcrypt

7. **Mongoose Documentation**
   https://mongoosejs.com/docs/

8. **JWT (JSON Web Token) Authentication Guide**
   https://jwt.io/introduction

9. **Axios HTTP Client**
   https://axios-http.com/docs/intro

10. **MDN Web Docs – JavaScript, HTML, and CSS**
    https://developer.mozilla.org/